Mapped Value = Diff Exp + 2

Diff Exp = Mapped Value - 2

In the D15 mode, the above mapping is applied to each individual differential exponent for coding into the bit stream. In the D25 mode, each *pair* of differential exponents is represented by a single mapped value in the bitstream. In this mode the second differential exponent of each pair is implied as a delta of 0 from the first element of the pair as indicated by the following table:

diff exp n	diff exp n+1	mapped value
+2	0	4
+1	0	3
0	0	2
-1	0	1
-2	0	0

The D45 mode is similar to the D25 mode except that *quads* of differential exponents are represented by a single mapped value, as indicated by the following table:

diff exp n	diff exp n+1	diff exp n+2	diff exp n+3	mapped value
+2	0	0	0	4
+1	0	0	0	3
0	0	0	0	2
-1	0	0	0	1
-2	0	0	0	0

Since a single exponent is effectively shared by 2 or 4 different mantissas, encoders must ensure that the exponent chosen for the pair or quad is the minimum absolute value (corresponding to the largest exponent) needed to represent all the mantissas.

For all modes, sets of three adjacent (in frequency) mapped values (M1,M2 and M3) are grouped together and coded as a 7 bit value according to the following formula:

Coded 7 bit Grouped Value = $(25 \times M1) + (5 \times M2) + M3$

The exponent field for a given channel in an AC-3 audio block consists of a number of these grouped values, preceded by a single absolute exponent.

3.3. Exponent Decoding

The exponent strategy for each coupled and independent channel is included in a set of 2 bit fields designated chexpstr[n]. When the coupling channel is present a cplexpstr field is also included. These fields are decoded as follows:

chexpstr[n], cplexpstr	exponent strategy
00	reuse prior exponents

01	D15
10	D25
11	D45

When the low frequency effects channel is enabled the lfeexpstr field is present. It is decoded as follows:

lfeexpstr	exponent strategy
0	reuse prior exponents
1	D15

Following the exponent strategy fields is a set of channel bandwidth codes, chbwcod[n]. These are only present for independent channels (channels not in coupling) that have new exponents in the current block. The channel bandwidth code defines the end mantissa bin number for that channel according to the following:

endmant[n] = ((chbwcod[n] + 12) * 3) + 37 (n=independent channel)

For coupled channels the end mantissa bin number is defined by the starting bin number of the coupling channel:

endmant[n] = cplstrtmant (n=coupled channel)

where cplstrtmant is as derived below. By definition the starting mantissa bin number for independent and coupled channels is 0.

strtmant[n] = 0

For the coupling channel, the frequency bandwidth information is derived from the fields cplbegf and cplendf found in the coupling strategy information. The coupling channel starting and ending mantissa bins are defined as:

cplstrtmant = (cplbegf * 12) + 37

cplendmant = ((cplendf + 3) * 12) + 37

The low frequency effects channel, when present, always starts in bin 0 and always has the same number of mantissas:

lfestrtmant = 0

lfeendmant = 7

The second set of fields contains coded exponents for all channels indicated to have new exponents in the current block. These fields are designated as exps[n][i] for independent and coupled channels, cplexps[i] for the coupling channel, and lfeexps[i] for the low frequency effects channel. The first element of each exps field (exps[n][0]) and the lfeexps field (lfeexps[0]) is always a 4 bit absolute number. For these channels the absolute exponent always contains the exponent value of the first transform coefficient (bin #0). The absolute exponent for the coupled channel, cplabsexp, is only used as a reference to begin decoding the differential exponents for the coupling channel (i.e. it does not represent an actual exponent). The cplabsexp is contained in the audio block as a 4 bit value, however it corresponds to a 5 bit value. The LSB of the coupled channel initial exponent is always 0, so the decoder must take the 4 bit value which was sent, and double it (left shift by 1) in order to obtain the 5 bit starting value.

For each coded exponent set the number of grouped exponents (not including the first absolute exponent) to decode from the bit stream is derived as follows:

For independent and coupled channels:

nchgrps[i]	= truncate ((endmant[i] - 1) / 3)	for D15 mode
	= truncate ((endmant[i] + 2) / 6)	for D25 mode
	= truncate ((endmant[i] + 5) / 12)	for D45 mode

For the coupling channel:

ncplgrps	= (cplendmant - cplstrtmant) / 3	for D15 mode
	= (cplendmant - cplstrtmant) / 6	for D25 mode
	= (cplendmant - cplstrtmant) / 12	for D45 mode

For the low frequency effects channel:

nlfegrps = 2

Decoding a set of coded grouped exponents will create a set of 5 bit absolute exponents. The exponents are decoded as follows:

- Each 7 bit grouping of mapped values (gexp) is decoded using the inverse of the encoding procedure:

  M1 = truncate (gexp / 25)

  M2 = truncate ((gexp % 25) / 5)

  M3 = (gexp % 25) % 5

- Each mapped value is converted to a differential exponent (dexp) by subtracting the mapping offset:

  dexp = M - 2

- The set of differential exponents if converted to absolute exponents by adding each differential exponent to the absolute exponent of the previous frequency bin:

  exp[n] = exp[n-1] + dexp[n]

- For the D25 and D45 modes each absolute exponent is copied to the remaining members of the pair or quad.

The above procedure can be summarized as follows:

```
for (i = 0; i < ngrps; i++)          /* unpack the mapped values */
{
    expacc = gexp[i]
    dexp[i * 3] = truncate (expacc / 25)
    expacc = expacc - ( 25 * dexp[i * 3])
    dexp[(i * 3) + 1] = truncate ( expacc / 5)
    expacc = expacc - (5 * dexp[(i * 3) + 1])
    dexp[(i * 3) + 2] = expacc
}
for (i = 0; i < (ngrps * 3); i++)          /* unbiased mapped values */
{
    dexp[i] = dexp[i] - 2
}
prevexp = absexp                           /* convert from differential to absolute */
for (i = 0; i < (ngrps * 3); i++)
{
    aexp[i] = prevexp + dexp[i]
    prevexp = aexp[i]
}
exp[0] = absexp
for (i = 0; i < (ngrps * 3); i++)          /* expand to full absolute exponent array */
{
    for (j = 0; j < grpsize; j++)
    {
        exp[(i * grpsize) + j +1] = aexp[i];
    }
}
```

where,

ngrps = number of grouped exponents (nchgrps[n], ncplgrps, or nlfegrps)

gsize = group size	= 1	for D15
	= 2	for D25
	= 4	for D45

absexp = absolute exponent (exps[n][0], (cplabsexp<<1), or lfeexps[0])

For the coupling channel the above output array, exp[n], should be offset to correspond to the coupling start mantissa bin:

cplexp[i + cplstrtmant] = exp[i + 1]

For the remaining channels exp[n] will correspond directly to the absolute exponent array for that channel.

4. BIT ALLOCATION

4.1. Overview

The bit allocation routine analyzes the spectral envelope of the audio signal being coded with respect to masking effects to determine the number of bits to assign to each transform coefficient mantissa. In the encoder, the bit allocation is performed globally on the ensemble of channels as an entity, from a common bit pool. There are no preassigned exponent or mantissa bits, allowing the routine to flexibly allocate bits across channels, frequencies, and audio blocks in accordance with signal demand.

The bit allocation contains a parametric model of human hearing for estimating a noise level threshold, expressed as a function of frequency, which separates audible from inaudible spectral components. Various parameters of the hearing model can be adjusted by the encoder depending upon signal characteristics. For example, a prototype masking curve is defined in terms of two piecewise continuous line segments, each with its own slope and y-axis intercept. One of several possible slopes and intercepts is selected by the encoder for each line segment. The encoder iterates on one or more such parameters until an optimal result is obtained. When all parameters used to estimate the noise level threshold have been selected by the encoder, the final bit allocation is computed. The model parameters are conveyed to the decoder with side information, which then executes the routine in a single pass.

The estimated noise level threshold is computed over 50 bands of non uniform bandwidth (an approximate 1/6 octave scale). The banding structure, defined by tables in the next section, is independent of sampling frequency. The required bit allocation for each mantissa is established by performing a table lookup based upon the difference between the input signal power spectral density (PSD), evaluated on a fine-grain uniform frequency scale, and the estimated noise level threshold, evaluated on the coarse-grain (banded) frequency scale. Therefore, the bit allocation result for a particular channel has spectral granularity corresponding to the exponent strategy employed. More specifically, a separate bit allocation will be computed for each mantissa within a D15 exponent set, each pair of mantissas within a D25 exponent set, and each quadruple of mantissas within a D45 exponent set.

The bit allocation must be computed in the decoder whenever the exponent strategy (chexpstr, cplexpstr, lfeexpstr) for one or more channels is not reused, or whenever baie, snroffste, or deltabaie = 1. Accordingly, the bit allocation can be updated at a rate ranging from once per audio block to once per 6 audio blocks, including the integral steps in between. A complete set of new bit allocation information is always transmitted in audio block 0.

Since the parametric bit allocation routine must generate identical results in all encoder and decoder implementations, each step is defined exactly in terms of fixed-point integer operations and table lookups. Throughout the discussion below, signed two's complement arithmetic is employed. All additions are performed with an accumulator of 14 or more bits.

4.2. Parametric Bit Allocation

This section describes the seven-step procedure for computing the output of the parametric bit allocation routine in the decoder. The approach outlined here starts with a single uncoupled or coupled exponent set and processes all the input data for each step prior to continuing to the next one. This technique, called vertical execution, is conceptually straightforward to describe and implement. Alternatively, the seven steps can be executed horizontally, in which case multiple passes through all seven steps are made for separate subsets of the input exponent set.

The choice of vertical vs. horizontal execution depends upon the relative importance of execution time vs. memory usage in the final implementation. Vertical execution of the algorithm is usually faster due to reduced looping and context save overhead. However, horizontal execution requires less RAM to store the temporary arrays generated in each step. Hybrid horizontal/vertical implementation approaches are also possible which combine the benefits of both techniques.

4.2.1. Initialization

Compute start/end frequencies for the channel being decoded. These are computed from parameters in the bitstream as follows:

```
endmant = 37 + (3 × (chbwcod + 12))    for D15, D25, D45 if exponents not coupled
        = 37 + (12 × cplbegf)               if exponents coupled
cplstrtmant = 37 + (12 × cplbegf)
cplendmant = 37 + (12 × (cplendf + 3))
```

The strtmant, lfestartmant, and lfeendmant parameters are known a priori to be 0, 0, and 7, respectively.

Compute fdecay, sdecay, fgain, sgain, and dbknee from parameters in the bitstream as follows:

```
fdecay = fastdec[fdcycod]
sdecay = slowdec[sdcycod]
sgain = slowgain[sgaincod]
dbknee = dbpbtab[dbpbcod]
```

Initialize as follows for main-channel uncoupled exponents:

```
start = strtmant
end = endmant
lowcomp = 0
fgain = fastgain[fgaincod]
snroffset = ((csnroffst – 15) << 2 + fsnroffst) << 3
```

Initialize as follows for coupled exponents:

```
start = cplstrtmant
end = cplendmant
fgain = fastgain[cplfgaincod]
snroffset = ((csnroffst – 15) << 2 + cplfsnroffst) << 3
if (cplleake)
{
    fastleak = (cplfleak << 8) + 512
    slowleak = (cplsleak << 8) + 512
}
```

Initialize as follows for low frequency channel exponents:

```
start = lfestrtmant
end = lfeendmant
lowcomp = 0
fgain = fastgain[lfefgaincod]
snroffset = ((csnroffst – 15) << 2 + lfefsnroffst) << 3
```

The tables for fastdec[] and slowdec[] are shown in Tables 4.1ab. The tables for fastgain[] and slowgain[] are shown in Tables 4.2ab. The table for dbpbtab[] is shown in Table 4.3.

4.2.2. Exponent Mapping into PSD

This step maps the channel exponents into a 13-bit signed log power-spectral density function.

```
for (i = start; i < end; i++)
{
    psd[i] = (3072 – (locexps[i] << 7))
}
```

Since locexps[k] assumes integral values ranging from 0 to 24, the dynamic range of the psd[] values is from 0 (for the lowest-level signal) to 3072 for the highest-level signal. The resulting function is represented on a fine-grain, linear frequency scale.

4.2.3. PSD Integration

This step of the algorithm integrates fine-grain PSD values within each of a multiplicity of 1/6th octave bands. The band structure is described in Tables 4.4 and 4.5. Table 4.4 contains the array values for bndtab[] and bndsz[], each of length 50 words. The bndtab[] array gives the first mantissa number in each band. The bndsz[] array provides the width in mantissas of each band. Table 4.5 contains the array values for masktab[], of length 256, providing the mapping from mantissa numbers into the associated 1/6 octave band number.

Note that Tables 4.4 and 4.5 contain redundant information, all of which need not be stored in an actual implementation. They are shown here for simplicity of presentation only.

The integration of PSD values in each band is performed with log-addition. The log-addition is implemented by computing the difference between the two operands and using the absolute difference divided by 2 as an address into a length 256 lookup table, latab[], shown in Table 4.6.

```
j = start
k = masktab[start]
do
{
    bndpsd[k] = psd[j]
    j++
    for (i = j; i < min(bndtab[k+1], end); i++)
    {
        bndpsd[k] = logadd(bndpsd[k], psd[j])
        j++
    }
    k++
}
while (end > bndtab[k++])
```

```
logadd(a, b)
{
    c = a – b
    address = max((abs(c) >> 1), 255)
    if (c >= 0)
    {
        return(a + latab(address))
    }
    else
    {
        return(b + latab(address))
    }
}
```

4.2.4. Compute Excitation Function

The excitation function is computed by applying the prototype masking curve selected by the encoder (and transmitted to the decoder) to the integrated PSD spectrum (bndpsd[]). The result of this computation is then offset downward in amplitude by the fgain and sgain parameters, which are also obtained from the bitstream.

```
bndstrt = masktab[start]
bndend = masktab[end – 1] + 1
if (bndstrt == 0)  /* For channel which is not coupled */
{  /* Note:  Do not call calc_lowcomp() for the last band of the lfe channel, (i = 6). */
    lowcomp = calc_lowcomp(lowcomp, bndpsd[0], bndpsd[1], 0)
    excite[0] = bndpsd[0] – fastgain –lowcomp
    lowcomp = calc_lowcomp(lowcomp, bndpsd[1], bndpsd[2], 1)
    excite[1] = bndpsd[1] –fastgain –lowcomp
    begin = 7
    for (i = 2; i < 7; i++)
    {
        lowcomp = calc_lowcomp(lowcomp, bndpsd[i], bndpsd[i+1], i)
        fastleak = bndpsd[i] –fastgain
        slowleak = bndpsd[i] –slowgain
        excite[i] = max(fastleak –lowcomp, slowleak)
        if (bndpsd[i] <= bndpsd[i+1])
        {
            begin = i + 1
            break
        }
    }
    for (i = begin; i < min(bndend, 22); i++)
    {
        lowcomp = calc_lowcomp(lowcomp, bndpsd[i], bndpsd[i+1], i)
```

```
            fastleak −= fastdec
            fastleak = max(fastleak, bndpsd[i] - fastgain)
            slowleak −= slowdec
            slowleak = max(slowleak, bndpsd[i] - slowgain)
            excite[i] = max(fastleak −lowcomp, slowleak)
        }
        begin = 22
    }
    else /* For channel which is coupled */
    {
        begin = bndstrt
    }
    for (i = begin; i < bndend; i++)
    {
        fastleak −= fastdec
        fastleak = max(fastleak, bndpsd[i] - fastgain)
        slowleak −= slowdec
        slowleak = max(slowleak, bndpsd[i] −slowgain)
        excite[i] = max(fastleak, slowleak)
    }
    calc_lowcomp(a, b0, b1, bndnum)
    {
        if (bndnum < 20)
        {
            if (b0 + 256 == b1)
            {
                a = min(384, a + 128)
            }
            else if (b0 > b1)
            {
                a = max(0, a − 128)
            }
        }
        else
        {
            a = max(0, a − 128)
        }
        return(a)
    }
```

4.2.5. Compute Masking Curve

This step computes the masking (noise level threshold) curve from the excitation function, as shown below. The hearing threshold table is shown in Table 4.7. The fscod and dbpbcod variables are received by the decoder in the bitstream.

```
dbknee = dbpbtab[dbpbcod]
for (i = bndstrt; i < bndend; i++)
{
    if (bndpsd[i] < dbknee)
    {
        excite[i] += ((dbknee – bndpsd[i]) >> 2)
    }
    mask[i] = max(excite[i], hth[fscod][i])
}
```

4.2.6. Apply Delta Bit Allocation

The optional delta bit allocation information in the bitstream provides a means for the encoder to transmit side information to the decoder which directly increases or decreases the masking curve obtained by the parametric routine. Delta bit allocation can be enabled by the encoder for audio blocks which derive an improvement in audio quality when the "default" bit allocation is appropriately modified. The delta bit allocation option is available for each full bandwidth channel and the coupled channel.

Modifications to the decoder bit allocation are transmitted as side information. The modifications occur in the form of adjustments to the default masking curve computed in the decoder. Adjustments can be made in multiples of ±6 dB. On the average, a masking curve adjustment of –6 dB corresponds to an increase of 1 bit of resolution for all the mantissas in the affected 1/6th octave band.

```
if ((deltbae == 0) || (deltbae == 1))
{
    band = 0
    for (i = 0; i < deltnseg; i++)
    {
        band += deltoffst[i]
        if (deltba[i] >= 4)
        {
            delta = (deltba[i] – 3) << 7
        }
        else
        {
            delta = (deltba[i] –4) << 7
        }
        for (k = 0; k < deltlen[i]; k++)
        {
            mask[band] += delta
            band++
        }
    }
}
```

4.2.7. Compute Bit Allocation

The bit allocation pointer array (bap[]) is computed in this step. The masking curve, adjusted by snroffset in an earlier step and then truncated, is subtracted from the fine-grain psd[] array. The difference is right-shifted by 5 bits, thresholded, and then used as an address into baptab[] to obtain the final allocation. The baptab[] array is shown in Table 4.8.

The sum of all channel mantissa allocations in one frame is constrained by the encoder to be less than or equal to the total number of mantissa bits available for that frame. The encoder accomplishes this by iterating on the values of csnroffst and fsnroffst (or cplfsnroffst or lfefsnroffst for coupled and low frequency effects channels) to obtain an appropriate result. Hence, in the absence of channel errors, the decoder is guaranteed to receive a mantissa allocation which meets the constraints of a fixed transmission bit-rate.

At the end of this step, the bap[] array contains a series of 4-bit pointers. The pointers indicate how many bits are assigned to each mantissa. The correspondence between bap pointer value and quantization accuracy is presented in Table 4.9.

```
i = start
j = bndtab[start]
floor = floortab[floorcod]
do
{
    mask[j] -= snroffset
    mask[j] -= floor
    if (mask[j] < 0)
    {
        mask[j] = 0
    }
    mask[j] &= 0x1fe0
    for (k = i; k < min(bndtab[j], end); k++)
    {
        address = (psd[i] - mask[j]) >> 5
        address = min(63, max(0, address))
        bap[i] = babtab[address]
        i++
    }
}
while (end > bndtab[j++])
```

4.3. Bit Allocation Tables

Table 4.1a Fast decay table (fastdec[])

address	fastdec
0	0x3f
1	0x53

2	0x67
3	0x7b

Table 4.1b Slow decay table (slowdec[])

address	slowdec
0	0x0f
1	0x11
2	0x13
3	0x15

Table 4.2a Fast gain table (fastgain[])

address	fastgain
0	0x080
1	0x100
2	0x180
3	0x200
4	0x280
5	0x300
6	0x380
7	0x400

Table 4.2b Slow gain table (slowgain[])

address	slowgain
0	0x540
1	0x4d8
2	0x478
3	0x410

Table 4.3 Knee of db/bit characteristic curve (dbpbtab[])

address	dbpbtab
0	0x000
1	0x700
2	0x900
3	0xb00

Table 4.4 Masking array floor value table (floortab[])

address	floortab

0	0x200
1	0x180
2	0x100
3	0x080

Table 4.5 First mantissa and band width of each 1/6th octave band, as a function of band number

(bndtab[] and bndsz[], respectively)

band #	bndtab	bndsz
0	0	1
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	1
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1

band #	bndtab	bndsz
25	25	1
26	26	1
27	27	1
28	28	3
29	31	3
30	34	3
31	37	3
32	40	3
33	43	3
34	46	3
35	49	6
36	55	6
37	61	6
38	67	6
39	73	6
40	79	6
41	85	12
42	97	12
43	109	12
44	121	12
45	133	24
46	157	24
47	181	24
48	205	24
49	229	24

Table 4.6 1/6th octave band number as a function of exponent number (masktab[])

The address is (10 × A) + B

	B=0	B=1	B=2	B=3	B=4	B=5	B=6	B=7	B=8	B=9
A=0	0	1	2	3	4	5	6	7	8	9

A=1	10	11	12	13	14	15	16	17	18	19
A=2	20	21	22	23	24	25	26	27	28	28
A=3	28	29	29	29	30	30	30	31	31	31
A=4	32	32	32	33	33	33	34	34	34	35
A=5	35	35	35	35	35	36	36	36	36	36
A=6	36	37	37	37	37	37	37	38	38	38
A=7	38	38	38	39	39	39	39	39	39	40
A=8	40	40	40	40	40	41	41	41	41	41
A=9	41	41	41	41	41	41	41	42	42	42
A=10	42	42	42	42	42	42	42	42	42	43
A=11	43	43	43	43	43	43	43	43	43	43
A=12	43	44	44	44	44	44	44	44	44	44
A=13	44	44	44	45	45	45	45	45	45	45
A=14	45	45	45	45	45	45	45	45	45	45
A=15	45	45	45	45	45	45	45	46	46	46
A=16	46	46	46	46	46	46	46	46	46	46
A=17	46	46	46	46	46	46	46	46	46	46
A=18	46	47	47	47	47	47	47	47	47	47
A=19	47	47	47	47	47	47	47	47	47	47
A=20	47	47	47	47	47	48	48	48	48	48
A=21	48	48	48	48	48	48	48	48	48	48
A=22	48	48	48	48	48	48	48	48	48	49
A=23	49	49	49	49	49	49	49	49	49	49
A=24	49	49	49	49	49	49	49	49	49	49
A=25	49	49	49	0	0	0				

Table 4.7 Log-addition table (latab[]) The address is (10 × A) + B

	B=0	B=1	B=2	B=3	B=4	B=5	B=6	B=7	B=8	B=9
A=0	0x0040	0x003f	0x003e	0x003d	0x003c	0x003b	0x003a	0x0039	0x0038	0x0037
A=1	0x0036	0x0035	0x0034	0x0034	0x0033	0x0032	0x0031	0x0030	0x002f	0x002f
A=2	0x002e	0x002d	0x002c	0x002c	0x002b	0x002a	0x0029	0x0029	0x0028	0x0027
A=3	0x0026	0x0026	0x0025	0x0024	0x0024	0x0023	0x0023	0x0022	0x0021	0x0021
A=4	0x0020	0x0020	0x001f	0x001e	0x001e	0x001d	0x001d	0x001c	0x001c	0x001b
A=5	0x001b	0x001a	0x001a	0x0019	0x0019	0x0018	0x0018	0x0017	0x0017	0x0016
A=6	0x0016	0x0015	0x0015	0x0015	0x0014	0x0014	0x0013	0x0013	0x0013	0x0012
A=7	0x0012	0x0012	0x0011	0x0011	0x0011	0x0010	0x0010	0x0010	0x000f	0x000f
A=8	0x000f	0x000e	0x000e	0x000e	0x000d	0x000d	0x000d	0x000d	0x000c	0x000c
A=9	0x000c	0x000c	0x000b	0x000b	0x000b	0x000b	0x000a	0x000a	0x000a	0x000a
A=10	0x000a	0x0009	0x0009	0x0009	0x0009	0x0009	0x0008	0x0008	0x0008	0x0008
A=11	0x0008	0x0008	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0006	0x0006
A=12	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0005	0x0005	0x0005	0x0005

A=13	0x0005	0x0005	0x0005	0x0005	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004
A=14	0x0004	0x0004	0x0004	0x0004	0x0004	0x0003	0x0003	0x0003	0x0003	0x0003
A=15	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0002
A=16	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
A=17	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0001	0x0001
A=18	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=19	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=20	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
A=21	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=22	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=23	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=24	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
A=25	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000

Table 4.8 Hearing threshold table for fs = 48, 44.1, 32 kHz (hth[fscod][])

band number	hth 48 k	hth 44.1 k	hth 32 k
0	0x04d0	0x04f0	0x0580
1	0x04d0	0x04f0	0x0580
2	0x0440	0x0460	0x04b0
3	0x0400	0x0410	0x0450
4	0x03e0	0x03e0	0x0420
5	0x03c0	0x03d0	0x03f0
6	0x03b0	0x03c0	0x03e0
7	0x03b0	0x03b0	0x03d0
8	0x03a0	0x03b0	0x03c0
9	0x03a0	0x03a0	0x03b0
10	0x03a0	0x03a0	0x03b0
11	0x03a0	0x03a0	0x03b0
12	0x03a0	0x03a0	0x03a0
13	0x0390	0x03a0	0x03a0
14	0x0390	0x0390	0x03a0
15	0x0390	0x0390	0x03a0
16	0x0380	0x0390	0x03a0
17	0x0380	0x0380	0x03a0
18	0x0370	0x0380	0x03a0
19	0x0370	0x0380	0x03a0
20	0x0360	0x0370	0x0390
21	0x0360	0x0370	0x0390
22	0x0350	0x0360	0x0390
23	0x0350	0x0360	0x0390

band number	hth 48 k	hth 44.1 k	hth 32 k
25	0x0340	0x0350	0x0380
26	0x0330	0x0340	0x0380
27	0x0320	0x0340	0x0370
28	0x0310	0x0320	0x0360
29	0x0300	0x0310	0x0350
30	0x02f0	0x0300	0x0340
31	0x02f0	0x02f0	0x0330
32	0x02f0	0x02f0	0x0320
33	0x02f0	0x02f0	0x0310
34	0x0300	0x02f0	0x0300
35	0x0310	0x0300	0x02f0
36	0x0340	0x0320	0x02f0
37	0x0390	0x0350	0x02f0
38	0x03e0	0x0390	0x0300
39	0x0420	0x03e0	0x0310
40	0x0460	0x0420	0x0330
41	0x0490	0x0450	0x0350
42	0x04a0	0x04a0	0x03c0
43	0x0460	0x0490	0x0410
44	0x0440	0x0460	0x0470
45	0x0440	0x0440	0x04a0
46	0x0520	0x0480	0x0460
47	0x0800	0x0630	0x0440
48	0x0840	0x0840	0x0450

24	0x0340	0x0350	0x0380
49	0x0840	0x0840	0x04e0

Table 4.9 Bit allocation pointer table (baptab[])

address	baptab
0	0
1	1
2	1
3	1
4	1
5	1
6	2
7	2
8	3
9	3
10	3
11	4
12	4
13	5
14	5
15	6
16	6
17	6
18	6
19	7
20	7
21	7
22	7
23	8
24	8
25	8
26	8
27	9
28	9
29	9
30	9
31	10
32	10
33	10
34	10
35	11
36	11
37	11
38	11
39	12
40	12
41	12
42	12
43	13
44	13
45	13
46	13
47	14
48	14
49	14
50	14
51	14
52	14
53	14
54	14
55	15
56	15
57	15
58	15
59	15
60	15
61	15
62	15
63	15

Table 4.10 Number of quantizer levels and bits per mantissaas a function of bap number

bap	quantizer levels	mantissa bits [group bits]

0	0	0
1	3	1.67 [5]
2	5	2.33 [7]
3	7	3
4	11	3.5 [7]
5	15	4
6	32	5
7	64	6
8	128	7
9	256	8
10	512	9
11	1024	10
12	2048	11
13	4096	12
14	16,384	14
15	65,536	16

5. QUANTIZATION AND DECODING OF MANTISSAS

5.1. Overview

Mantissas are encoded using three techniques: symmetric quantization, asymmetric quantization, and grouping. All mantissas are quantized. Those with a small number of quantizer levels use symmetric quantization. All others use asymmetric quantization. Asymmetric quantization is a conventional two's complement quantization. Grouping is used to conserve bits for some symmetrically quantized mantissas. Two or three mantissas are compositely coded into one bitstream word.

In the encoder, each transform coefficient is left justified by shifting it left the number of times indicated by its exponent. Once this is done the shifted transform coefficient words are quantized to correspond to the number of quantizer levels defined by the bit allocation pointers, bap[k]. Note that k represents the transform coefficient frequency index; k = 0, 1, 2, ...*max*, where $max \leq 252$ for the 512 point transformation.

The following table indicates which quantizer to use for each bap. If a bap equals 0, no bits are sent for the mantissa. Instead, it will be replaced by shaped noise in the decoder. Grouping is used for baps of 1, 2 and 4 (3, 5, and 11 level quantizers.)

bap	quantizer levels	quantization type	mantissa bits [group bits]
0	0	none	0
1	3	symmetric	1.67 [5]
2	5	symmetric	2.33 [7]
3	7	symmetric	3
4	11	symmetric	3.5 [7]
5	15	symmetric	4
6	32	asymmetric	5
7	64	asymmetric	6
8	128	asymmetric	7
9	256	asymmetric	8
10	512	asymmetric	9
11	1024	asymmetric	10
12	2048	asymmetric	11
13	4096	asymmetric	12
14	16,384	asymmetric	14
15	65,536	asymmetric	16

During the decode process, the mantissa data stream is parsed up into single mantissas of varying length, interspersed with groups representing combined coding of either triplets or pairs of mantissas. In the bitstream, the mantissas in each exponent set are arranged in frequency ascending order. However, groups occur at the position of the first mantissa contained in the group. Nothing is unpacked from the bit stream for the subsequent mantissas in the group.

5.2. Expansion of Mantissas for Asymmetric Quantization

For bit allocation pointer array values, $6 \leq \text{bap} \leq 15$, asymmetric fractional two's complement quantization is used. Each mantissa, along with its exponent, are the floating point representation of a transform coefficient. The decimal point is considered to be to the left of the MSB; therefore the mantissa word represents the range of $[1.0 - 2^{-\text{qntztab[bap[k]]}}]$ to -1.0. The mantissa of length qntztab[bap[k]] is extracted from the bit stream. Conversion back to a fixed point representation is achieved by right shifting the mantissa by its exponent. This process is represented by the following formula:

transform_coefficient[k] = mantissa[k] >> exponent[k]

No grouping is done for asymmetrically quantized mantissas. See the section on dither for a discussion of handling mantissas with zero bits.

5.3. Expansion of Mantissas for Symmetrical Quantization

Symmetrically quantized mantissas are converted to standard fractional 2's complement binary words by a table lookup. The number of bits indicated by a mantissa's bap are extracted from the bit stream and right justified. These values are treated as integer codes. Each code is used as an index to look up the floating point mantissa to use. The resulting values are right shifted by the corresponding exponents to generate the transform coefficient values.

bap = 1 (qntz3lev)

mantissa code	mantissa value
0	-2./3
1	0
2	2./3

bap = 2 (qntz5lev)

mantissa code	mantissa value
0	-4./5
1	-2./5
2	0
3	2./5
4	4./5

bap = 3 (qntz7lev)

mantissa code	mantissa value
0	-6./7
1	-4./7
2	-2./7
3	0
4	2./7
5	4./7
6	6./7

bap = 4 (qntz11lev)

mantissa code	mantissa value
0	-10./11
1	-8./11
2	-6./11
3	-4./11
4	-2./11
5	0
6	2./11
7	4./11
8	6./11
9	8./11

10	10./11

bap[k] = 5 (qntz15lev)

mantissa code	mantissa value
0	-14./15
1	-12./15
2	-10./15
3	-8./15
4	-6./15
5	-4./15
6	-2./15
7	0
8	2./15
9	4./15
10	6./15
11	8./15
12	10./15
13	12./15
14	14./15

transform_coefficient[k] = quantization_table[mantissa_code[k]] >> exponent[k]

5.4. Ungrouping of Mantissas

In the case when bap = 1, 2, or 4, the resulting data words are further compressed by combining 3 level words and 5 level words into separate groups representing triplets of mantissas, and 11 level words into groups representing pairs of mantissas. Groups are filled in the order that the mantissas are processed. If the number of mantissas in an exponent set does not fill an integral number of groups, the groups are shared across exponent sets. The next exponent set in the block continues filling the partial groups. If the total number of 3 or 5 level quantized transform coefficient derived words are not each divisible by 3, or if the 11 level words are not divisible by 2, the final groups of a block are padded with dummy mantissas to complete the composite group. Dummies should be ignored by the decoder. Groups are extracted from the bitstream using the length derived from bap. Three level quantized mantissas (bap = 1) are grouped into triples each of 5 bits. Five level quantized mantissas (bap = 2) are grouped into triples each of 7 bits. Eleven level quantized mantissas (bap = 4) are grouped into pairs each of 7 bits.

Encoder equations

bap = 1:

group_code = 9 * mantissa_code[a] + 3 * mantissa_code[b] + mantissa_code[c]

bap = 2:

group_code = 25 * mantissa_code[a] + 5 * mantissa_code[b] + mantissa_code[c]

bap = 4:

group_code = 11 * mantissa_code[a] + mantissa_code[b]

Decoder equations

bap = 1:

mantissa_code[a] = truncate (group_code / 9)
mantissa_code[b] = truncate ((group_code % 9) / 3)
mantissa_code[c] = (group_code % 9) % 3

bap = 2:

mantissa_code[a] = truncate (group_code / 25)
mantissa_code[b] = truncate ((group_code % 25) / 5)
mantissa_code[c] = (group_code % 25) % 5

bap = 4:

mantissa_code[a] = truncate (group_code / 11)
mantissa_code[b] = group_code % 11
where mantissa a comes before mantissa b, which comes before mantissa c

5.5. Dither for Zero Bit Mantissas

The AC-3 decoder uses random noise (dither) values instead of quantized values when the number of bits allocated to a mantissa is zero (bap = 0). The use of the random value is conditional on the value of dithflag. When the value of dithflag is 1, the random noise value is used. When the value of dithflag is 0, a true zero value is used. There is a dithflag variable for each channel. Dither is applied after the individual channels are extracted from the coupling channel. In this way, the dither applied to each channel's upper frequencies is uncorrelated.

Any reasonably random sequence may be used to generate the dither values. The word length of the dither values is not critical. Eight bits is sufficient. The optimum scaling for the dither words is to take a uniform distribution of values between -1 and +1, and scale this by 0.707, resulting in a uniform distribution between +0.707 and - 0.707. A scalar of 0.75 is close enough to also be considered optimum. A scalar of 0.5 (uniform distribution between +0.5 and -0.5) is acceptable, but will not yield optimum results.

Once a dither value is assigned to a mantissa, the mantissa is right shifted according to its exponent to generate the corresponding transform coefficient.

transform_coefficient[k] = scaled_dither_value >> exponent[k]

6. CHANNEL COUPLING

6.1. Overview

If enabled, channel coupling is performed on encode by averaging the transform coefficients across channels that are included in the coupling channel. Each coupled channel has a unique coupling coordinate array which is used to preserve the high frequency envelope of the five original channels. The coupling process is performed above a coupling frequency that is defined by the cplbegf word.

The decoder converts the coupling channel to the 5 original channels by multiplying the coupled channel transform coefficient values by 8 times the appropriate coupling coordinate for a particular channel and frequency sub-band. An additional processing step occurs for the 2/0 mode. If the phsflginu bit = 1 or the equivalent state is continued from a previous block, then phase restoration bits are sent in the bit stream via phase flag bits. The phase flag bits represent the coupling sub-bands in a frequency ascending order. If a phase flag bit = 1 for a particular sub-band, all the right channel transform coefficients within that coupled sub-band are negated after modification by the coupling coordinate, but before inverse transformation.

6.2. Subband Structure For Coupling

512 sample block: Transform coefficients # 37 through 252 are grouped into 18 subbands of 12 coefficients each.

sb #	low tc #	high tc #	lf cutoff (kHz) @ fs=48 kHz	hf cutoff (kHz) @ fs=48 kHz	lf cutoff (kHz) @ fs=44.1 kHz	hf cutoff (kHz) @ fs=44.1 kHz
0	37	48	3.42	4.55	3.14	4.18
1	49	60	4.55	5.67	4.18	5.21
2	61	72	5.67	6.80	5.21	6.24
3	73	84	6.80	7.92	6.24	7.28
4	85	96	7.92	9.05	7.28	8.31
5	97	108	9.05	10.17	8.31	9.35
6	109	120	10.17	11.30	9.35	10.38
7	121	132	11.30	12.42	10.38	11.41
8	133	144	12.42	13.55	11.41	12.45
9	145	156	13.55	14.67	12.45	13.48
10	157	168	14.67	15.80	13.48	14.51
11	169	180	15.80	16.92	14.51	15.55
12	181	192	16.92	18.05	15.55	16.58
13	193	204	18.05	19.17	16.58	17.61
14	205	216	19.17	20.30	17.61	18.65
15	217	228	20.30	21.42	18.65	19.68
16	229	240	21.42	22.55	19.68	20.71
17	241	252	22.55	23.67	20.71	21.75

256 sample block: Transform coefficients # 19 through 126 are grouped into 18 subbands of 6 coefficients each.

sb #	low tc #	high tc #	lf cutoff (kHz) @ fs=48 kHz	hf cutoff (kHz) @ fs=48 kHz	lf cutoff (kHz) @ fs=44.1 kHz	hf cutoff (kHz) @ fs=44.1 kHz
0	19	24	3.47	4.59	3.19	4.22
1	25	30	4.59	5.72	4.22	5.25
2	31	36	5.72	6.84	5.25	6.29
3	37	42	6.84	7.97	6.29	7.32
4	43	48	7.97	9.09	7.32	8.35
5	49	54	9.09	10.22	8.35	9.39
6	55	60	10.22	11.34	9.39	10.42
7	61	66	11.34	12.47	10.42	11.46
8	67	72	12.47	13.59	11.46	12.49
9	73	78	13.59	14.72	12.49	13.52
10	79	84	14.72	15.84	13.52	14.56
11	85	90	15.84	16.97	14.56	15.59
12	91	96	16.97	18.09	15.59	16.62
13	97	102	18.09	19.22	16.62	17.66
14	103	108	19.22	20.34	17.66	18.69
15	109	114	20.34	21.47	18.69	19.72
16	115	120	21.47	22.59	19.72	20.76
17	121	126	22.59	23.72	20.76	21.79

Note: At 32 kHz sampling rate the subband frequency ranges are 2/3 the values of those for 48 kHz.

6.3. Coupling Coordinate Format

A coupling coordinate is sent in the data stream as a floating point number, specified with a 4-bit exponent and a 4-bit mantissa. For each coupled channel, a 2-bit master coupling coordinate is used to gain range all of the coupling coordinates within that channel.

Computation of the coupling coordinate from the bitstream elements is as follows (assuming channel n, subband i):

```
if (cplcoexp[n, i] == 15)
{
    coupling_coordinate_mantissa[n, i] = cplco[n, i] / 16
}
else
{
    coupling_coordinate_mantissa[n, i] = (16 + cplco[n, i]) / 32
}

coupling_coordinate_exponent[n, i] = cplcoexp[n, i] + 3*mstrcplco[n]

coupling_coordinate[n, i] = coupling_coordinate_mantissa[n, i] >> coupling_coordinate_exponent[n, i]
```

Individual channel mantissas are then reconstructed from the coupled channel as follows (assuming 512 sample block):

```
for (j = 0 to 11)
{
    chmant[n, i*12 + j + 37] = cplmant[i*12 + j + 37] * coupling_coordinate[n, i] * 8
}
```

With 4-bit exponents, 4-bit mantissas, and 2-bit master coupling coordinates, the total scale factor dynamic range representable is 114 dB, (-110 dB to +4 dB), with step sizes varying between 0.28 and 0.56 dB.

7. REMATRIXING

7.1. Overview

Rematrixing in AC-3 is a channel combining technique in which sums and differences of highly correlated channels are coded rather than the original channels themselves. That is, rather than code and pack left and right in a two channel coder, we construct:

left' = 0.5*(left + right)

right' = 0.5*(left - right)

We then perform the usual quantization and data packing operations on left' and right'. Clearly, if the original stereo signal were identical in both channels (i.e. two-channel mono), this technique will result in a left' signal that is identical to the original left and right channels, and a right' signal that is identically zero. As a result, we can code the right' channel with very few bits, and increase accuracy in the more important left' channel.

This technique is especially important for preserving Dolby Surround compatibility. To see this, consider a two channel mono source signal such as that described above. A Dolby Pro-Logic decoder will try to steer all in-phase information to the center channel, and all out-of-phase information to the surround channel. If rematrixing is not active, the Pro-Logic decoder will receive the following signals:

Received left = left + QN1

Received right = right + QN2

where QN1 and QN2 are independent (i.e. uncorrelated) quantization noise sequences, which correspond to the AC-3 coding algorithm quantization and are program dependent. The Pro-Logic decoder will then construct center and surround channels as:

center = 0.5*(left + QN1) + 0.5*(right + QN2)

surround = 0.5*(left + QN1) - 0.5*(right + QN2)   (ignoring the 90 degree phase shift)

In the case of the center channel, QN1 and QN2 add, but remain masked by the dominant signal left + right. In the surround channel, however, left - right cancels to zero, and the surround speakers are left to reproduce the difference in the quantization noise sequences (QN1 - QN2).

If channel rematrixing is active, the center and surround channels will be more easily reproduced as:

center = left' + QN1

surround = right' + QN2

In this case, the quantization noise in the surround channel QN2 is much lower in level, as it is masked by the difference signal right'.

7.2. Frequency Band Definitions

In AC-3, rematrixing is performed independently in separate frequency bands. There are four bands with boundary locations equal to:

If coupling not used (cplinu == 0)

nrematbd = 4

512 sample block

			fs = 48 kHz	fs = 48 kHz	fs = 44.1 kHz	fs = 44.1 kHz
band #	low coeff #	high coeff #	low freq (kHz)	high freq (kHz)	low freq (kHz)	high freq (kHz)
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	252	5.67	23.67	5.21	21.75

256 sample block

			fs = 48 kHz	fs = 48 kHz	fs = 44.1 kHz	fs = 44.1 kHz
band #	low coeff #	high coeff #	low freq (kHz)	high freq (kHz)	low freq (kHz)	high freq (kHz)
0	7	12	1.22	2.34	1.12	2.15
1	13	18	2.34	3.47	2.15	3.19
2	19	30	3.47	5.72	3.19	5.25
3	31	126	5.72	23.72	5.25	21.79

if coupling is used (cplinu == 1) and cplbegf > 2

nrematbd = 4

512 sample block

			fs = 48 kHz	fs = 48 kHz	fs = 44.1 kHz	fs = 44.1 kHz
band #	low coeff #	high coeff #	low freq (kHz)	high freq (kHz)	low freq (kHz)	high freq (kHz)
0	13	24	1.17	2.30	1.08	2.11
1	25	36	2.30	3.42	2.11	3.14
2	37	60	3.42	5.67	3.14	5.21
3	61	A	5.67	B	5.21	C